VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Detekce vozidel v leteckých snímcích

Vehicle Detection in Aerial Images

Jan Moták

Bakalářská práce

Vedoucí práce: Ing. Michael Holuša, Ph.D.

Ostrava, 2021

Abstrakt

Bakalářská práce se zabývá implementací aplikace pro detekci vozidel v leteckých snímcích. Tato aplikace může být uplatněna v dopravní bezpečnosti pro detekci kolon a nehod, nebo s asistenčními prvky jako jsou informace o obsazenosti parkovišť a hustoty provozu. Pro výslednou aplikaci byla zvolena architektura YOLO, která je založena na technologii neuronových sítí a jejich schopnostech učit se poznávat vozidla pomocí trénování sítě z vytvořeného datasetu. Bakalářská práce kromě aplikace rozebírá tématiku potřebnou k pochopení zvoleného řešení a výsledné porovnání různých architektur na zadaném problému.

Klíčová slova

Neuronové sítě; Detekce; YOLO

Abstract

This bachelor's thesis is about the implementation of an application for vehicle detection in aerial images. This application can be used in traffic safety for traffic jams and accident detection or in assistance elements such as information about the availability of spots in parking lots or density of traffic. The final application is based on YOLO architecture which is based on a neural network and its ability to learn from training with a custom dataset. Bachelor's thesis besides application explains theory which is needed to understand the topic and finally a comparison of different architectures on the given problem.

Keywords

Neural networks; Detection; YOLO

Poděkování

Zde bych rád poděkoval všem, kteří mi pomohli k úspěšnému dokončení této práce, zejména mému vedoucímu panu Ing. Michaelu Holušovi, Ph.D. za cenné rady a ochotu při psání této práce.

Obsah

Seznam použitých zkratek a symbolů

NN	–	Neuronová sít
CNN	–	Konvoluční neuronová sít
SSD	–	Single-shot detektor
YOLO	–	You Only Look Once
TP	–	True Positive
FN	–	False Negative
FP	–	False Positive
mAP	–	Mean Average Precision
AP	–	Average Precision
IoU	–	Intersection over Union
NMS	–	Non-maximum Suppresion
GPU	–	Grafický procesor

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Neuronové sítě a algoritmy na nich založené jsou využívány čím dál tím více a v posledních letech jejich vývoj význačně vzrostl. Cílem práce je implementovat aplikaci a popsat teorii potřebnou k pochopení problematiky. Tato aplikace bude umožňovat detekci vozidel z leteckých snímků pro čtyři kategorie: auto, nákladní auto, motorka a autobus. Detekce vozidel může být využita například při detekci volných míst na parkovištích, detekci kolon nebo vyhledání vozidel z těžko dostupných míst. Aplikace bude pro detekci využívat architekturu YOLO neboli You Only Look Once a následně budou získané výsledky porovnány s jinými architekturami. Dále v práci budou popsány problematické části detekce vozidel z leteckých snímků a porovnání na těchto částech. Samotný text práce obsahuje 3 hlavní kapitoly. Cílem kapitoly 2 je vysvětlit teorii potřebnou k pochopení funkčnosti detektorů, vysvětlit problematiku detekce vozidel z leteckých snímků a popsat architektury, které je možno využít pro jejich implementaci. Kapitola 3 popisuje hlouběji zvolenou architekturu a její verzi YOLOv5. Dále tato kapitola obsahuje informace o zvolených datasetech a prostředcích pro vytvoření datasetu a detektoru. Na závěru této kapitoly budou popsány výsledky natrénovaného detektoru a popsány problematické části a vysvětleno jejich řešení. V poslední kapitole 4 budou následně porovnány výsledky vytvořeného detektoru s výsledky, které byly získány volbou jiných trénovacích parametrů, jiných parametrů detektoru, jiné verze datasetu nebo jiné architektury.

Kapitola 2

Úvod do problematiky

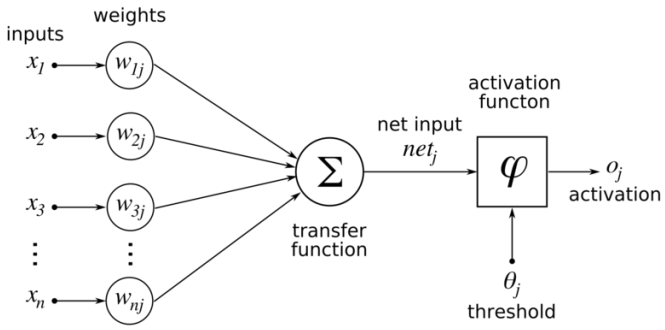
Detekce vozidel z leteckých snímků je problematika, která se skládá z několika kroků a používá některé technologie, které je vhodné nejdříve vysvětlit. Z tohoto důvodu je tedy před ukázkami výsledků detektoru a popsáním implementace vhodné vysvětlit teorii, která je s problematikou detekce v obraze spojena. Dále v této kapitole bude popsána problematika detekce vozidel z leteckých snímků a architektury, které na vytvoření detektoru mohou být použity.

2.1 Neuronové sítě

Neuronové sítě jsou inspirované funkcí nervového systému v lidském nebo zvířecím mozku a jeho schopnostmi jako jsou učení, předpovědi a rozpoznávání vzorců. Podobně jako u nervového systému člověka se jedná se o síť skládající se z neuronů, které jsou mezi sebou vzájemně propojeny spoji (synapsemi) a tyto spoje jsou ohodnoceny váhami, což umožňuje neuronům zasílat signály. Tato spojení a schopnost učení se na základě trénovacího vstupu dává neuronovým sítím velkou možnost širšího uplatnění. Cílem neuronové sítě je tedy osvojit si schopnost vyprodukování výsledku z předem neznámého vstupu pomocí vybavování si informací z dat, které do sítě byly při jejím trénování předány.

Neuronové sítě mohou být využity k řešení mnoha problémů. Patří mezi ně například: Analýza zvuku (převody slov ve zvuku do písemné podoby)[1], Analýza obrazu (detekce vozidel na obrázku)[2] nebo například předpověď vývoje cen[3].

Jak již bylo předem zmíněno NN se skládají z neuronů, které jsou vzájemně propojeny a mohou si tak zasílat signály, které jsou reprezentovány celými čísly. Pro tyto neurony platí, že jsou schopny mít více vstupů s určitou váhou, která určuje sílu signálu, ale jen jeden výstup, který může být zaslán více neuronům. Dále neurony před výstupem obsahují aktivační funkci, což je matematická funkce, která určuje výstup neuronové sítě. Jejím cílem je zajistit nelineárnost výstupu neuronové sítě. Na obrázku 2.1 je možno vidět architekturu neuronu v NN.



Obrázek 2.1: Model neuronu [4]

NN jsou tvořeny vrstvami, které v sobě uchovávají neurony. V normálních NN existují tři typy vrstev a to vstupní, skryté a výstupní. Vstupní vrstva obsahuje a zpracovává data určená pro natrénování sítě. Skryté vrstvy tato data přijímají a snaží se z nich učit na základě jejich specifických rysů. Cílem výstupní vrstvy je vyprodukovat výstup vyhovující problematice, na kterou je síť trénována. Podle vzájemného propojení neuronů a typu aktivační funkce je rozlišováno více typů architektur neuronových sítí.

Pro natrénování sítí existuje více způsobů. Mezi první patří učení s učitelem, při kterém dochází k tomu, že síti jsou na vstupu zaslána data a čeká se na výsledek sítě. Pokud výsledek není správný, síti je zaslána negativní zpětná vazba a síť podle toho upraví své parametry. Při tomto typu trénování musí být vstupní data síti zasílána velice často, až do té doby, kdy bude síť schopna vyprodukovat výsledky odpovídající předpokladům s požadovanou úspěšností. Dalším typem trénování sítí je učení bez učitele. U tohoto typu trénování síť nezná požadovaný výsledek a jejím úkolem je provádět shlukovou analýzu a výsledky tak rozřadit do více tříd na základě jejich podobnosti, což umožní vytvoření vlastní reprezentace těchto objektů. [5][6]

2.2 Problematika detekce vozidel v leteckých snímcích

U detekce vozidel z leteckých snímků je důležité se zamyslet nad problémy, které mohou při detekci nastat. Mezi ně může patřit například: velikost vozidel v leteckých snímcích, podobnost vozidel vůči jiným objektům, vzdálenost leteckých snímků, vzdálenosti vozidel od sebe, denní doby a další. Z toho důvodu je důležité vybrat takovou architekturu, která bude vyhovovat našim požadavkům a vozidla detekovat s co největší přesností.

2.3 Detekce v obraze

Jedná se o techniku počítačového vidění, která umožňuje ve snímcích nebo ve videích lokalizovat a klasifikovat vyhledávané objekty. Pro detekci v obraze se nejčastěji využívají přístupy založené na strojovém učení. Pro tuto práci byla zvolena možnost detekce pomocí hlubokého učení, která je

typicky založena na konvolučních neuronových sítích (CNN). Před vysvětlením detekce samotné je vhodné nejprve vysvětlit dva pojmy, které detekce spojuje, a to klasifikaci a lokalizaci.

Klasifikace – používá se, pokud se ptáme, co se na daném snímku nachází. Cílem této techniky je tedy přiřadit snímku nějakou kategorii a na tuto otázku tak odpovědět. Jako omezení oproti detekci v obraze je však to, že klasifikace určuje pouze to, co se na snímku nachází, nikoli kde se to nachází.

Lokalizace – cílem lokalizace je odpovědět na otázku, kde na snímku se nějaký objekt nachází. Tedy určit jeho polohu a tu na snímku vyznačit. Výstupem je tedy obrázek, který obsahuje označení okolo objektu, který považuje za nejvíce zřejmý, ale jeho kategorie je mu neznámá.

Cílem detekce v obraze je spojení těchto dvou vlastností a vytvoření detektoru, který bude schopen na snímcích či videu lokalizovat, klasifikovat a ohodnotit vyhledávané objekty více kategorií a zbavit se tak jednotlivých nedostatků těchto dvou technik. Detekce v obraze má velmi důležité uplatnění v mnoha oblastech. Jako příklad využití detekce ve světě může tedy patřit například detekce tváří, počítání objektů nebo jako součást procesu autonomního řízení vozidel. [7]

2.3.1 Možnosti řešení

Pro problematiku detekce v obraze pomocí hlubokého učení existuje mnoho přístupu. Mezi nejznámější patří například RetinaNet [8], YOLO (You Only Look Once) [9], SSD (Single Shot Detector) [10], Region Proposal metody jako R-CNN [11], Fast-RCNN [12], Faster R-CNN [13] a další.

Jak již bylo dříve zmíněno, pro vytvoření detektoru v obraze existuje mnoho přístupu a je tedy vhodné ty nejrelevantnější hlouběji vysvětlit. Pro naše potřeby byla zvolena architektura YOLOv5, která bude podrobněji popsána v kapitole 3.

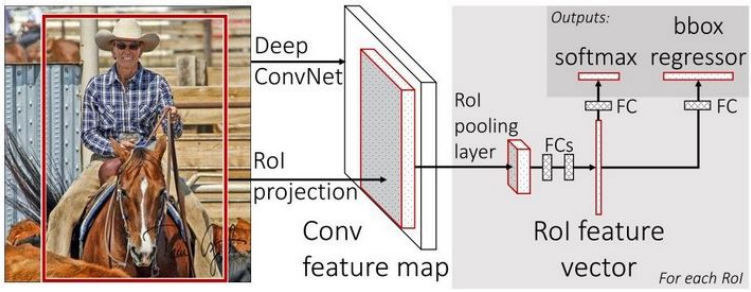
2.3.2 R-CNN

První verze R-CNN [11] vyšla roku 2014 za účelem řešení problému s efektivitou lokalizace při detekci. Předchozí řešení využívala pro generaci návrhů na regiony techniku zvanou exhaustive search, která využívá plovoucího okna různých velikostí právě za účelem vytvoření těchto návrhů na regiony. Exhaustive search je zde však nahrazen technikou selective search, která využívá výhody segmentace v objektech a exhaustive search k účinnému vytvoření návrhů na regiony. Selective search posléze vygeneruje návrhy na regiony a ty následně posílá do CNN modelu. Ten pro každý z těchto návrhů vygeneruje vektor rysů, který je následně zaslán do SVM modelu pro výslednou klasifikaci a lokalizaci.

Mezi velké nevýhody této architektury patří její rychlost a náročnost na potřebné místo na disku.

2.3.3 Fast R-CNN

Architektura Fast R-CNN [12] nahradila již dříve existující architekturu R-CNN a opravila tak několik nedostatků, které existovaly u architektury R-CNN. Největší změnou u této architektury je přidání ROI pooling vrstvy za účelem extrakce vektorů rysů pevné velikosti pro každý z návrhů na regiony. Další velkou změnou u architektury je to, že se skládá jen z jedné části oproti předchozí architektuře, která se skládala ze tří. Tuto architekturu je možno vidět podrobněji popsanou na obrázku 2.2.



Obrázek 2.2: Architektura Fast R-CNN [12]

Princip této architektury je velice podobný architektuře R-CNN, ale namísto toho, aby do sítě byly vkládány návrhy na regiony je do sítě zaslán celý snímek a z něj je vygenerována mapa rysů. Tato mapa rysů je následně zaslána do ROI pooling vrstvy za účelem extrakce vektorů pevné velikosti ze všech návrhů na regiony. Získané vektory jsou následně zaslány do takzvaných plně propojených vrstev a poté do vrstvy Softmax za účelem určení třídy návrhu. Největší nevýhodou této architektury je využití algoritmu Selective Search, který je poměrně pomalý.

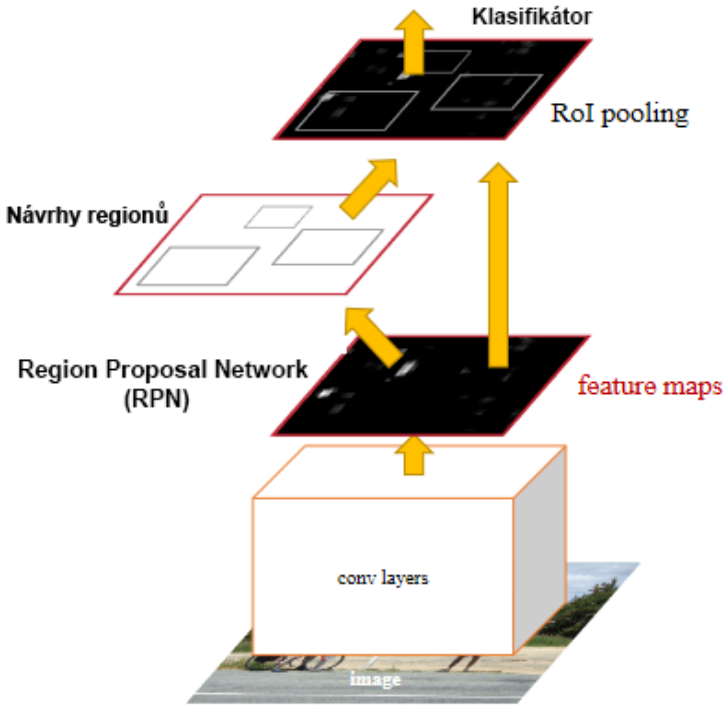
2.3.4 Faster R-CNN

Faster R-CNN je rozšíření již existující architektury Fast R-CNN a jak z názvu vyplývá, jedná se o výpočtově rychlejší architekturu. Tato architektura je rychlejší než Fast R-CNN především z důvodu přidání RPN neboli region proposal network. Dalším důležitým rozšířením od předchůdců z této rodiny je nahrazení konceptu pyramid snímků novým konceptem pevného označení, který bude později vysvětlen v kapitole 3.1.3. Nejdůležitější změnou je však to, že výpočty konvolučních vrstev se dělí mezi RPN a Fast R-CNN, což právě umožňuje redukci času potřebného na výpočty. Z tohoto popisu je tedy jasné, že Faster R-CNN se skládá z dvou hlavních částí a to:

- **RPN** – sloužící k vytvoření návrhů na regiony
- **Fast R-CNN** – jehož cílem je v právě vytvořených regionech detekovat vyhledávané objekty

Hlubším cílem RPN je zavedení konceptu zdůrazňování v neuronových sítích, neboli navádění detektoru Fast R-CNN v jakých oblastech na snímcích má detektor objekty vyhledávat. Na obrázku

2.3 lze vidět architekturu Faster R-CNN, ze které je zřejmé dělení složitosti výpočtu mezi RPN a moduly Faster R-CNN.



Obrázek 2.3: Architektura Faster R-CNN [13]

Funkčnost Faster R-CNN je tedy následující:

- Vygenerování návrhů na regiony pomocí RPN
- Pro každý z navržených regionů je získán vektor rysů pevné velikosti za pomocí RoI pooling vrstvy
- Získané vektory rysů jsou nadále poslány do Fast R-CNN, která je klasifikuje
- Jsou vráceny výsledné přesnosti a označení nalezených objektů

Kapitola 3

Implementace

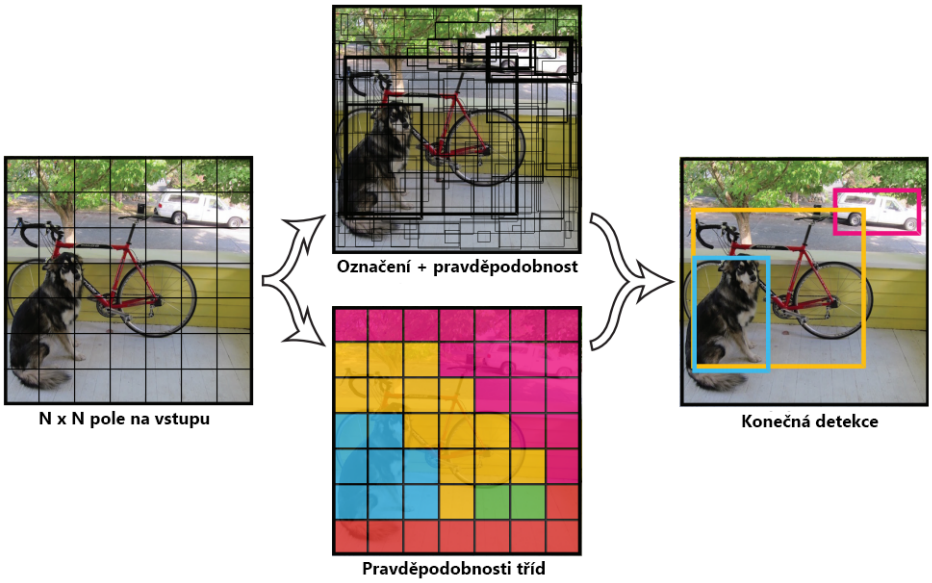
Jak bylo v předchozí kapitole zmíněno, pro řešení problematiky detekce vozidel z leteckých snímků existuje více architektur. Je tedy důležité zvolit takovou, která bude vozidla detekovat s co největší přesností a za rozumnou dobu, která byla k vytvoření detekcí potřeba. Z tohoto důvodu byla zvolena architektura You Only Look Once neboli YOLO, která v porovnání s jinými architekturami dosahuje srovnatelných přesností. Mezi největší výhody této architektury patří její rychlost. YOLO umožňuje rychlou a poměrně přesnou detekci a povoluje tak využití detektoru ve videích a živých přenosech. Po porovnání různých verzí těchto architektur byla zvolena verze YOLOv5, která je poměrně nová, ale poskytuje rychlejší trénování modelu a větší přesnost oproti jiným verzím této architektury.

3.1 YOLO:You Only Look Once

YOLO [9] patří mezi jednu z nejznámějších architektur pro detekci objektů v obraze. YOLO, podobně jako SSD se řadí mezi jedno fázové detektory. Princip této architektury je takový, že vstupní obrázek je zpracován konvoluční sítí pouze jednou, na rozdíl od jiných algoritmů, jako například Faster R-CNN [13], které vytváří návrhy na regiony a na ty síť aplikuje. Právě díky tomu se jedná o jednu z nejrychlejších architektur, která je schopna dosahovat dobrých výsledků při detekcích na videích v reálném čase. Mezi nevýhody této architektury patří nižší přesnost výsledného modelu, ta je však kompenzována vyšší výslednou rychlostí. Tento problém je však s každou novou verzí této architektury zlepšován.

Pro vytvoření finálních detekcí dochází po načtení vstupního snímku k jeho následnému rozdělení na NxN pole. Nad takto rozděleným snímkem dojde k aplikaci neuronové sítě, čímž dojde k vytvoření označení objektů (bounding box), jejich vah a dále k předpovědí tříd, ke kterým se nalezené objekty řadí (class probability). Výsledkem této operace je mnoho kandidátních označení, které jsou pomocí post-processing kroků upevněny do finální předpovědi. Pro získání finálních předpovědí se využívají dva klíčové post-processing kroky, a to Intersection Over Union a Non-maximum suppression. Právě tyto kroky jsou na snímky s kandidátními označeními aplikovány, a tak dochází k

odfiltrování nežádoucích označení a vytvoření finální předpovědi. Ukázku vytvoření finálních detekcí ze vstupního snímku lze vidět na obrázku 3.1



Obrázek 3.1: Proces vytváření detekcí [14]

Na obrázku výše lze v první části vidět vytvoření NxN pole nad vstupním obrázkem. V dalším kroku po aplikaci sítě dochází k vytvoření prozatímních označení včetně jejich vah pravděpodobnosti a také pravděpodobnosti tříd na buňku ve vytvořeném poli. V posledním kroku lze na obrázku vidět aplikaci předem zmiňovaných post-processing kroků pro vytvoření finálních detekcí a jejich zařazení do tříd.

Model architektury YOLO, se jako každý jiný jedno fázový detektor skládá ze tří hlavních částí, a to backbone (páteř), head (hlava) a neck (krk).

Backbone – jedná se o bázi klasifikačního modelu, na které je detekce objektů postavena. Skládá se primárně z konvolučních vrstev a jejím cílem je získání důležitých rysů ze vstupního obrázku

Neck – cílem této části modelu je vytvořit takzvané pyramidy rysů neboli feature pyramids, které slouží k tomu, aby byl model schopen dobře detekovat objekty různých velikostí

Head – část detektoru, která je zodpovědná za finální detekci a klasifikaci objektů. Tato část přijímá rysy (features), které byly vyprodukovány krkem detektoru, aplikuje na ně pevné označení a generuje tak finální výstupové vektory obsahující informace o detekovaných objektech

U této architektury je také vhodné popsat funkce, které se používají při procesu trénování sítě. Mezi tyto funkce patří aktivační, optimalizační a ztrátová funkce, jejichž vysvětlení je popsáno níže.

Aktivační funkce – jedná se o matematickou funkci, která pomáhá síti pochopit a naučit se komplexní vzory v datech. Mezi její další povinnosti patří rozhodovat o tom, co bude předáno do dalšího neuronu jako jeho vstup

Optimalizační funkce – cílem optimalizačních funkcí v neuronových sítích je změna parametrů neuronových sítí, jako jsou váhy a míra učení (learning rate) za účelem dosažení co nejmenších ztrát (loss)

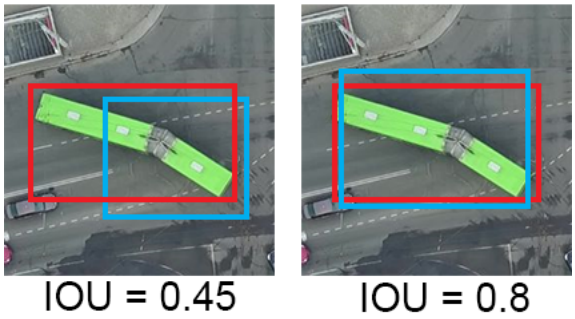
Ztrátová funkce – účelem ztrátové funkce je ohodnotit naši neuronovou síť nad daným problémem. Modely se tedy snaží ztrátu (loss) snížit na co nejmenší hodnotu, protože čím menší jsou ztráty, tím s větší přesností náš detektor lokalizuje a klasifikuje objekty. Je tedy velice důležité zvolit vhodnou ztrátovou funkci

3.1.1 Intersection over Union

Intersection over Union neboli Jaccard index je metrika, která umožňuje určit míru překrytí dvou označení. IoU je důležitá hodnota, jak pro trénování sítě, kde slouží pro porovnání vytvořeného označení a označení z datasetu, tak pro následnou detekci, kde je použito v Non Maximum Suppresion pro zvolení nejlepšího kandidáta na výsledné označení. Pro výpočet hodnoty IoU se využívá poměrně jednoduchý vzorec 3.1, jehož výstupem je hodnota z intervalu <0,1> určující výslednou míru překrytí těchto dvou označení.

$$IoU = \frac{Průnik\ oblastí}{Sjednocení\ oblastí} \tag{3.1}$$

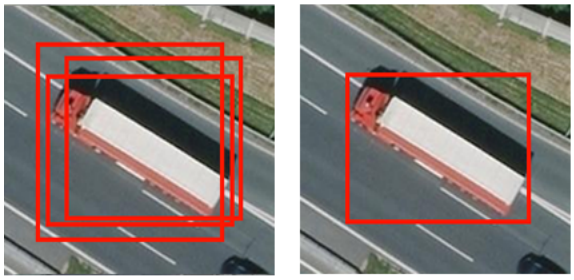
Na obrázku 3.2 lze vidět vypočtené hodnoty IoU na reálném případě, kdy červený obdélník zobrazuje označení z datasetu a modrý obdélník označení, které bylo získáno za použití detektoru.



Obrázek 3.2: Ukázka IoU na reálném příkladě

3.1.2 Non-maximum Suppression

NMS je jednou z nezbytných částí algoritmu YOLO. Používá se při detekci kvůli tomu, že při procházení vstupního obrázku detektorem dochází k tomu, že detektor označí vyhledávaný objekt vícekrát, je tedy velice důležité zvolit takové označení, které nejlépe označuje daný objekt. Cílem NMS je tedy to, aby odstranilo nežádoucí označení, tedy ty s menší pravděpodobností a zanechala jen ta, která nejlépe označují vyhledávané objekty. Princip NMS je tedy takový, že v prvním kroku odstraní označení s malou pravděpodobností a zanechá pouze ta s vyšší, než je požadovaná. Poté jsou vybrána označení s největší pravděpodobností a odstraněna ta s vysokou hodnotou IoU, tedy překrytí (většinou IoU ≥ 0.5). Tyto kroky jsou poté prováděny iterativně dokud nezbudou ohraničení, která lze smazat. Na obrázku 3.3 lze vidět zjednodušenou vizualizaci práce NMS, kdy v levém obrázku vidíme ohraničení před aplikací NMS a v pravém ohraničení po aplikaci NMS.



Obrázek 3.3: Ukázka NMS na reálném případě

3.1.3 Pevné označení

Pevná označení, neboli anchor boxy, jsou velice důležitou součástí nejenom detektorů rodiny YOLO, ale i architektur jako Faster RCNN, SSD nebo RetinaNet. Kdysi k detekci byla používána technika plovoucího okna, která procházela obrázek a na každou získanou část aplikovala klasifikátor. Brzy však bylo zjištěno, že toto řešení není efektivní, a tak bylo nahrazeno konvolučními sítěmi (CNN). Výstupem konvoluční sítě je pole (grid) rysů, které se skládá z buněk stejné velikosti. Toto pole má tedy buňky čtvercového tvaru, ale v případě, kdy vyhledávaný objekt není čtvercového tvaru dané velikosti, může nastat problém. Právě proto jsou zaváděna pevná označení. U takto rozděleného obrázku na NxN pole, každá z buněk obsahuje určitý počet pevných označení. Každé z těchto označení obsahuje vlastnosti jako souřadnice (x,y), výšku a šířku boxu, dále hodnotu objektové skóre, která udává míru toho s jakou pravděpodobností se objekt na tomto místě nachází, a nakonec pravděpodobnosti u všech tříd pro zjištění o jakou třídu z datasetu se jedná (třídní skóre). Za pomoci těchto boxů je tedy možné zvolit co nejpřesnější ohraničení objektů zvolením pevného označení s největším objektovým skóre a zkontrolovat o jakou třídu z datasetu se jedná za pomoci třídního skóre.

3.2 YOLOv5

YOLOv5 [15] je poměrně nový přírůstek do detektorů rodiny YOLO, jehož první verze vyšla 9. června 2020 a přináší nové dodatky do již existující implementace YOLOv3. YOLOv5 nabízí 4 hlavní typy konfigurací modelu, a to yolov5s, yolov5m, yolov5l a yolov5x, kdy každá má své výhody a nevýhody. Mezi hlavní rozdíly těchto konfigurací patří to, že menší váhy jako yolov5s nebo yolov5m umožňují rychlejší proces trénování a menší velikost konečného souboru na úkor konečné přesnosti modelu na rozdíl od konfigurací modelů yolov5l a yolov5x, které primárně upřednostňují konečnou přesnost modelu. V tabulce 3.1 lze vidět oficiální porovnání těchto konfigurací, které bylo provedeno na datasetu COCO.

Tabulka 3.1: Oficiální porovnání různých konfigurací architektury YOLOv5 [15]

Název konfigurace	$mAP_{0.5}$	$mAP_{0.5:0.95}$	Rychlost	FPS
YOLOv5s	0.556	0.368	**2.2ms**	**455**
YOLOv5m	0.631	0.445	2.9ms	345
YOLOv5l	0.664	0.481	3.8ms	264
YOLOv5x	**0.687**	**0.501**	6.0ms	167

Jak již bylo zmíněno v kapitole 3.1 model architektury YOLO se skládá ze tří hlavních částí. Jako páteř modelu YOLOv5 je využíván CSPNet [16] za účelem zisku důležitých rysů ze vstupního obrázku. Cílem CSPNet je obohatit schopnost sítě učit se takovým způsobem, aby výsledný model měl stejnou přesnost a při tom měl menší velikost. Mezi další rozšíření je zmenšení náročnosti na paměť zařízení. Jako krk modelu byla zvolena architektura PANet [17], hlavně kvůli její rychlosti a lepším možnostem přenosu dat mezi různými vrstvami modelu. Hlava modelu se od YOLOv3 a YOLOv4 nezměnila a zůstává stejná.

V předchozích verzích bylo nutné velikosti pevných označení nastavovat sám. YOLOv5 však aplikuje vlastní schopnost určení velikosti pevných označení, kde při průchodu datasetem jsou vypočítány jejich nejvhodnější velikosti.

3.2.1 Aktivační funkce

YOLOv5 využívá tři aktivační funkce, a to Sigmoid-Weighted Linear Units (SiLU) a HardWish, které jsou umístěny ve střední/skryté vrstvě, a funkci Sigmoid, která je umístěna ve finální detekční vrstvě.

Výstupem funkce Sigmoid je vždy číslo mezi 0 a 1. Mezi její nedostatky patří to, že v závěrečných hodnotách se hodnota Y nezvedá tak moc v závislosti s hodnotou X (zero gradients), to může zapříčinit zpomalení procesu trénování sítě. Na vzorci níže lze vidět výpočet této aktivační funkce.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

3.2.2 Optimalizační funkce

Pro optimalizaci YOLOv5 nabízí dvě takové funkce a to SGD a Adam, kdy od základu je používaná funkce SGD, ale je umožněno trénování s optimalizační funkcí Adam.

3.2.3 Ztrátová funkce

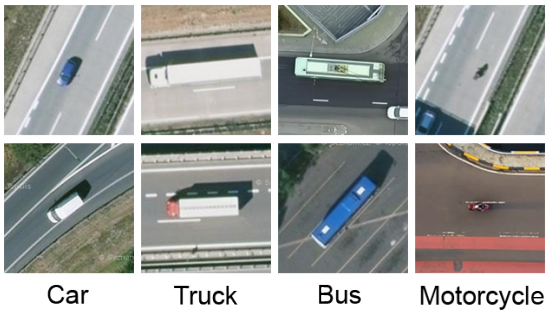
YOLOv5 využívá ke své činnosti dvě ztrátové funkce, a to Binary Cross-Entropy with Logits Loss a Focal Loss. Binary Cross-Entropy with Logits Loss je funkce, která kombinuje vrstvu Sigmoid a funkci Binary Cross-Entropy do jedné vrstvy pro docílení lepší stability. Focal loss je funkce, která je oproti BCE méně přísná a umožňuje tak potvrzování u objektů u kterých si detektor není na 80 - 100% jistý. Toto může být využito u nevybalancovaných datasetů a u detekcí objektů u kterých se jedná z velké části o pozadí objektů a z menší části o vyhledávaný objekt.

3.3 Dataset

Aby byla možná detekce v obraze jednou z nedílných částí je vytvoření datové sady neboli datasetu. Dataset jako takový obsahuje snímky s vyhledávanými objekty a k nim příslušně označené reprezentace hledaných objektů neboli anotace. Existuje velké množství volně dostupných a placených datasetů, kde každý má různé typy uplatnění. Mezi známé datasety, které je vhodné zmínit patří například COCO [18], který se často využívá pro srovnání přesnosti různých architektur.

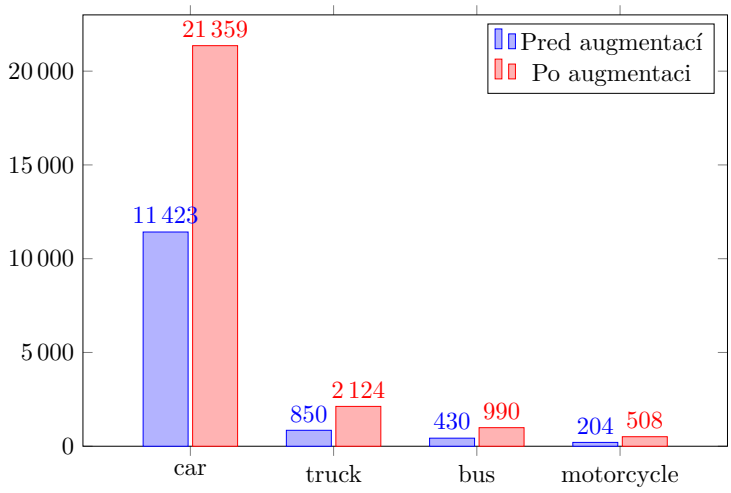
3.3.1 Vlastní dataset

Za účelem detekce vozidel z leteckých snímků byl k natrénování modelu vytvořen vlastní dataset. Při tvorbě datasetu vzniklo několik verzí, které budou později popsány a porovnány v kapitole 4.1. Pro veškeré výsledky byla použita jeho finální 4. verze, která bude podrobněji popsána níže. Finální verze datasetu obsahuje 1781 anotovaných leteckých snímků, kde byly označeny 4 typy objektů (tříd), a to car, truck, bus a motorcycle. Ukázky toho, jak vypadají reprezentace těchto tříd jsou zobrazeny na obrázku 3.4, který zobrazuje různé typy vozidel spadajících do těchto tříd.



Obrázek 3.4: Ukázka vozidel považovaných za třídy

V datasetu se nacházejí snímky z různých výšek z rozmezí 50 až 600 metrů. Data byly pořízeny z různých zdrojů, primárně z webového serveru *mapy.cz* a zbytek byl doplněn snímky z videí z webového serveru *pexels.com*. Pro lepší přesnost výsledného detektoru je vhodné, aby dataset obsahoval různorodá data. Z tohoto důvodu byla na dataset aplikována augmentace (jas ± 20%, saturace ± 25% a zrcadlení) a tím se zvýšila velikost datasetu z předchozích 891 anotovaných snímků na finálních 1781 anotovaných snímků. Na obrázku 3.5 lze vidět celkový počet jednotlivých reprezentací tříd v datasetu jak před aplikací augmentace, tak i po ní.



Obrázek 3.5: Reprezentace tříd v datasetu

Pro následné využití datasetu je nutné ho rozdělit do tří částí a to *train* pro proces trénování, *validation* pro kontrolu výsledných hodnot při trénování modelu a *test* na závěrečné odzkoušení natrénovaného modelu. Dataset byl do těchto částí rozdělen v poměru 70%, 20% a 10%.

3.3.2 LabelImg

Pro vytvoření datasetu byla použita aplikace LabelImg [19], což je grafický program, který umožňuje anotaci snímku a vytvoření datasetu. Program dovoluje anotace více tříd a následný export do dvou souborových formátů, a to YOLO text formátu a PascalVOC XML formátu. Z důvodu využití architektury YOLO, byl pro export využit YOLO text formát, který je jako jediný s touto architekturou kompatibilní

Program po následném exportu datasetu vytvoří soubor *classes.txt* obsahující třídy, které se v datasetu nacházejí. Dále je ke každému snímku vytvořen textový nebo XML soubor, který obsahuje informace o pozicích a třídách označených objektů.

3.3.3 Roboflow

Na úpravu a vylepšení datasetu byl použit webový server *app.roboflow.com*, pomocí kterého byly provedeny úpravy nad snímky v datasetu. Tyto úpravy byly provedeny především pro jeho zvětšení a vytvoření modifikovaných snímků například snížením jasu pro lepší detekci tmavších objektů. Další důležitou funkcí tohoto webového serveru je možnost exportu datasetu do jiných souborových formátu pro možnost porovnání jiných architektur.

3.4 Použité technologie

V této kapitole budou popsány technologie a knihovny, které byly použity pro implementaci detektoru.

3.4.1 Python

Pro implementaci byl zvolen programovací jazyk Python [20] verze 3.8.5, který je vedle C++ jedním z nejvíce využívaných programovacích jazyků pro vývoj aplikací založených na neuronových sítích.

3.4.2 PyTorch

PyTorch [21] je volně dostupná knihovna, která byla primárně vyvinuta firmou Facebook. Mezi dva hlavní účely této knihovny patří vytvoření balíčku umožňující implementaci neuronových sítí a nahrazení NumPy, za účelem využití GPU.

3.4.3 CUDA Toolkit 10.2

Výchozím prostředkem pro trénování sítě a detekci v obraze je procesor. Ten však má velké limitace při využití v problémech jako jsou detekce v obraze. Za účelem zrychlení těchto částí byl použit CUDA Toolkit [22], který umožňuje výrazně zrychlit výpočty aplikace využitím grafických karet značky Nvidia.

3.4.4 NVIDIA cuDNN

NVIDIA CUDA® Deep Neural Network library [23], neboli cuDNN je GPU zesílená knihovna, která obsahuje základní funkce pro neuronové sítě a poskytuje tak velice kvalitní a rychlé implementace pro často používané funkce. Další výhodou je zrychlení běžně používaných frameworků pro neuronové sítě jako TensorFlow nebo PyTorch.

3.5 Trénování sítě

Trénování modelu detektoru je poměrně časově náročná záležitost a může trvat od několika hodin až po dny. Z tohoto důvodu je velmi důležité pečlivě nastavit parametry trénování za účelem dosažení co nejlepších výsledků. Mezi tyto parametry trénování patří:

počet epoch – jedna epocha označuje průchod datasetu neuronovou sítí. Jelikož může být jedná epocha příliš velká, existuje následující pojem batch, jenž rozděluje dataset pro epochu na více částí

batch – jedná se o hodnotu toho, na kolik částí se mají obrázky z trénovací množiny rozdělit, pro rychlejší trénování a proto, aby proces trénování nebyl tak náročný na požadovanou paměť zařízeni

img-size – tento parametr rozhoduje o tom, na jakou velikost jsou snímky pro trénování a validaci přeměněny. Tento parametr může být důležitý v závislosti na velikosti vyhledávaných objektů, kdy při zvolení příliš malé nebo velké hodnoty může dojít k výraznému porušení specifických rysu objektů na snímcích

cfg – určení konfigurace modelu trénování (yolov5s, yolov5m, yolov5l, yolov5x)

optimalizační funkce – jak již bylo předem zmíněno YOLOv5 nabízí dvě optimalizační funkce, kde právě při procesu trénování je nutno jednu z nich zvolit

Výsledné parametry jsou zobrazeny v tabulce 3.2. Tyto parametry byly zvoleny po porovnání přesnosti výsledného modelu s ostatními, pro jejichž trénování byly zvoleny odlišné parametry.

Tabulka 3.2: Zvolené hodnoty trénování

batch	velikost snímku	počet epoch	konfigurace modelu	optimalizační funkce
4	1024px	500	yolov5x	SGD

Kromě těchto parametrů bylo nastaveno 5 pevných označení, jejichž velikosti byly vygenerovány průchodu datasetem. U detekce vozidel z leteckých snímku, kdy se většinou setkáváme se objekty malých rozměrů a různých kvalit, bylo jedním z nejdůležitějších parametrů nastavení img-size na 1024px. Po porovnání závěrečných modelů na různých konfiguracích byla nakonec zvolena konfigurace yolov5x, která označovala vozidla s největší přesností. Jako poslední parametr, a to optimalizační funkce, byla zvolena funkce SGD, která v porovnání s optimalizační funkci Adam pracovala s větší přesností a stabilitou. Tento problém a výsledky využití obou těchto optimalizačních funkcí budou porovnány v kapitole 4.4.

3.6 Výsledné hodnoty trénování

Po dokončení trénování je nutné výsledný model ohodnotit a zjistit, zdali výsledky odpovídají našim požadavkům. Je tedy nutné určit, kde detektor fungoval dobře a naopak kde ne. V tomto smyslu existují tři pojmy, které popisují výsledky vzniklých detekcí modelu. Mezi tyto pojmy patří True Positive (TP) a False Positive (FP). Pro určení toho, co je TP nebo FP je nutno nastavit hraniční hodnotu IoU, která je primárně nastavena na 0.5. Za TP jsou považována taková označení, kde IoU mezi detekcí a ručně vytvořeným označením je větší, než je hraniční hodnota IoU, což znamená že se na obrázku objekt nachází a my jsme ho dokázali správně lokalizovat. V případě, kdy hodnota IoU je menší než je nastavená hraniční hodnota, se jedná o FP, tedy vytvořená detekce označila nesprávný objekt. Posledním pojmem je False Negative (FN), což je hodnota, která udává počet nenalezených objektů, které náš detektor nebyl schopen lokalizovat. Tyto pojmy mají dále další využití pro výpočet výsledných hodnot a následné ohodnocení modelu.

3.6.1 Precision a recall

Za jedny z hlavních výsledných hodnot lze považovat precision a recall. Precision udává hodnotu toho, s jakou přesností model označuje objekty v testovacích obrázcích jako pozitivní, neboli nám udává hodnotu toho, kolik z označených objektů jsou správně označeny. Z tohoto vyplývá i vzorec 3.3, na kterém je vidět, že hodnota precision se vypočítá jako poměr všech správných detekcí a všech detekcí, které detektor vytvořil.

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)} \tag{3.3}$$

Recall je hodnota, která určuje to, jak dobře model vytváří predikce. Jedná se tedy o hodnotu udávající, kolik objektů byl detektor schopen nalézt vůči všem, které se na snímku nacházejí. Na vzorci 3.4 lze vidět, že se jedná o velice podobný vzorec, jako v předchozím případě, vzorec však obsahuje hodnotu FN namísto hodnoty FP.

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)} \tag{3.4}$$

V porovnání těchto dvou parametrů lze tedy usoudit, že pokud má model vysoký recall a nízkou precision, tak ohodnocuje většinu pozitivních případů správně, ale kromě toho označuje i objekty, které nejsou vyhledávanými. Naopak, když model má vysokou precision a nízký recall, tak dochází k tomu, že model nedělá zbytečné chyby při označování objektů, ale velkou část vyhledávaných objektů, které se na snímku nacházejí schopen nalézt nebyl. [24]

3.6.2 Mean average precision

Další výslednou hodnotou, kterou je nutno zmínit je mAP neboli mean Average Precision, což je jedna z metrik, která slouží k závěrečnému ohodnocení modelu. Před vysvětlením mAP je důležité zmínit Average Precision (AP), což je také metrika, která nám umožňuje shrnout precision-recall křivku do jedné výsledné hodnoty. Jedna z možností jak vypočíst AP, je vypočítat oblast pod precision-recall křivkou, tento výpočet je popsán na vzorci 3.5.

$$AP = \int_0^1 p(r)dr \tag{3.5}$$

Tento výpočet je však pro potřeby detekce v obraze nedostatečný, a proto se používá způsob, kdy se hodnota AP počítá z vyhlazené precision-recall křivky. Vyhlazení této křivky probíhá následujícím způsobem, kdy v případě toho, že hodnota precision klesá, je vždy volena největší hodnota napravo od místa poklesu. Jeden z možných způsobů, jak vypočíst hodnotu AP pro potřeby detekce, je rozdělit hodnoty recall na vyhlazené precision-recall křivce na N stejných částí. Výsledné AP je pak vypočítáno pomocí vzorce 3.6, na kterém lze vidět vyhlazování křivky, o což se stará funkce $p_{interp}(r)$ a následný výpočet hodnoty AP jako průměr hodnot precision na N hodnotách recall.

$$AP = \frac{1}{N} \sum_{n=1}^{N} p_{interp}(r) \tag{3.6}$$

Poté, co byla vysvětlena hodnota AP, lze říci, že mAP je hodnota, která byla získána pomocí průměru všech AP pro každou ze tříd, které se v datasetu nachází. Výsledný výpočet hodnoty mAP je pak popsán na vzorci 3.7, kde N udává počet tříd, které se v datasetu nacházejí, a AP_k určuje hodnotu AP pro třídu k. [25]

$$mAP = \frac{1}{N} \sum_{k=1}^{N} AP_k \tag{3.7}$$

Tato hodnota se objevuje ve více verzích. Mezi dvě hlavní verze této hodnoty patří $mAP_{0.5}$ a $mAP_{0.5:0.95}$, kde hodnota dolního indexu označuje, na jakém IoU byla výsledná hodnota mAP měřena. Tedy v $mAP_{0.5}$ byla hodnota IoU nastavena na 0.5 a u $mAP_{0.5:0.95}$ se jedná o průměr všech hodnot měřených na IoU z intervalu 0.5 až 0.95 s krokem 0.05.

3.6.3 F1-skóre

Pro ohodnocení modelu je vhodné zvolit více metrik. Další a poslední výslednou hodnotou je F1-score. Tato hodnota udává rovnováhu mezi hodnotami precision a recall. Čím větší jsou tedy rozdíly mezi hodnotami precision a recall, tím nižší je hodnota F1-skóre. [24]

Tato skutečnost je zobrazena na vzorci 3.8, který popisuje výpočet hodnoty F1-skóre.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.8}$$

3.7 Výsledky trénování

Po dokončení procesu trénování byl model otestován na základní testovací sadě a byly získány výsledky popisující jeho úspěšnost. Při prvním pohledu na tabulku 3.3 lze vidět, že hodnoty precision a recall pro všechny třídy jsou poměrně blízko sebe, ale model se přiklání k tomu, že některé objekty není schopen správně lokalizovat a klasifikovat.

Tabulka 3.3: Výsledky trénování modelu na testovací sadě

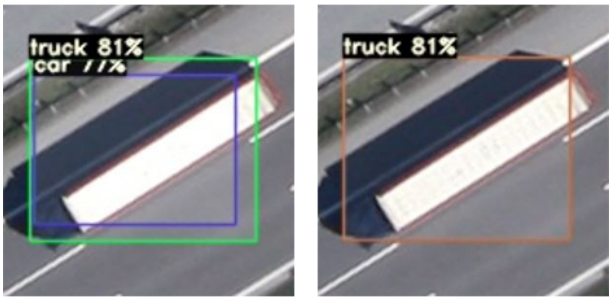
Název třídy	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre
všechny třídy	0.943	0.842	0.907	0.737	0.889
bus	0.917	0.798	0.912	0.803	0.853
car	**0.989**	**0.908**	**0.953**	0.726	**0.947**
motorcycle	0.953	0.758	0.829	0.584	0.844
truck	0.913	0.905	0.934	**0.835**	0.909

Při testování modelu je výchozí hodnota hranice přesnosti často nastavena na 0.001 především z důvodu pečlivého otestování přesnosti detektoru. U detekcí je však velice důležité nastavit vhodnou hodnotu hranice přesnosti. Tedy nastavit takovou hodnotu hranice přesnosti, která odfiltruje špatně označené objekty, a přitom bude stále dobře označovat ty správně označené. U nastavování této hodnoty je velice důležité se zamyslet nad tím, k čemu je detektor ve výsledku použít. Například, pokud je detektor využíván v lékařství, nebo pro detekci u autonomního řízení vozidel je vhodnější, aby detektor více označoval i ty špatné shody, tedy měl vyšší recall. Za účelem dosažení co nejlepšího odfiltrování nežádoucích objektů byla výchozí hodnota hranice přesnosti nastavena na 0.75. Ukázku detekce při nastavení různé hranice přesnosti lze vidět na obrázků 3.6, kdy pro obrázek vlevo byla hranice přesnosti nastavena na výchozí hodnotu a pro obrázek vpravo byla tato hranice nastavena na 0.001.

Obrázek 3.6: Ukázka rozdílů nastavení různé hranice přesnosti pro detektor

V tabulce 3.3 jsou dále vidět jednotlivé výsledné hodnoty pro všechny třídy, které se v datasetu nacházejí. Zde lze vidět, že největších hodnot $mAP_{0.5}$ dosáhla třída car a nejmenších hodnot mAP třída motorcycle, což je způsobeno těžkým rozpoznáváním motocyklů z velké výšky.

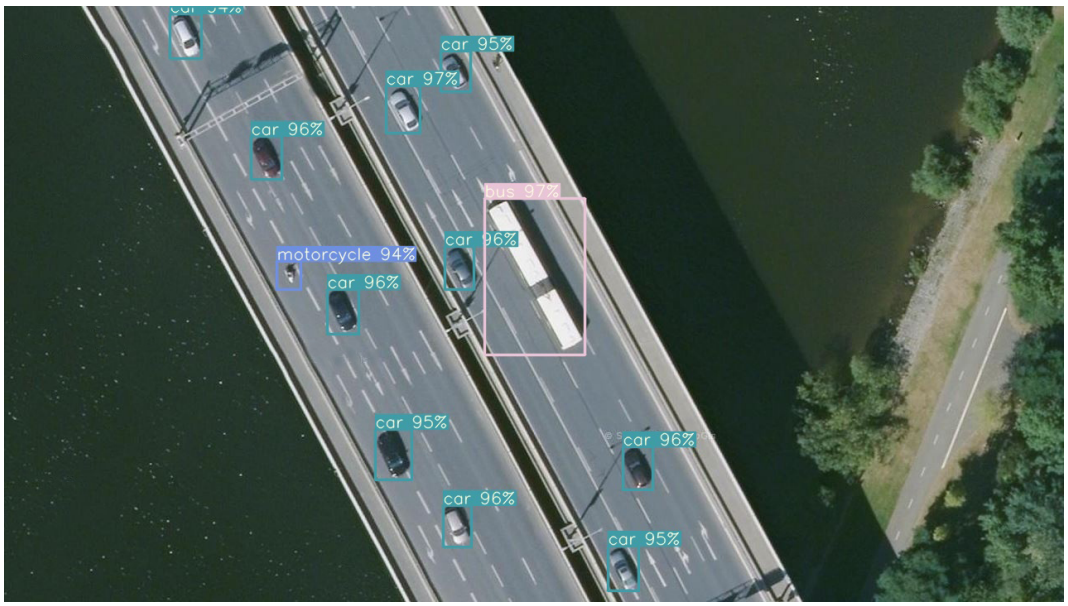
Mezi další důležitý faktor detektoru patří aplikování NMS mezi všemi třídami. Ukázku aplikace této funkce lze vidět na obrázku 3.7, kde na levém obrázku je vidět detekce bez aplikace NMS mezí třídami a na pravém po aplikaci této funkce.



Obrázek 3.7: Ukázka aplikace funkce NMS mezi třídami

Toto je velice důležité, protože objekty jako autobus či nákladní vozidlo může detektor z leteckého snímku označit dvakrát, jednou jakou autobus a podruhé jako nákladní vozidlo. Z tohoto důvodu je třeba aplikovat NMS pro všechny třídy, aby detektor vymazal označení s menší hodnotu pravděpodobnosti.

Poslední velice důležitou funkcí detektoru je nastavení toho, na jakou velikost má detektor obrázky změnit. Nastavení této hodnoty je velice závislé na rozlišení vstupního snímku. Pokud je tedy do detektoru vkládán obrázek s vysokým rozlišením, je velice důležité nastavit velikost změny vstupního obrázku na hodnotu, která umožní dobrou detekci vozidel i po změně jeho velikosti. Ukázku detekce po aplikaci těchto funkcí je možno vidět na obrázku 3.8.



Obrázek 3.8: Ukázka detekce na vybraném snímku z testovací množiny

Na obrázků lze vidět, že detektor byl schopen všechny vozidla úspěšně lokalizovat a klasifikovat. Jedná se však o ideální případ a detektor nemůže vždy detekovat vozidla s takovou přesností.

3.8 Problémové části detekce

Při vyzkoušení detektoru na několika testovacích snímcích bylo zjištěno, že detektor označuje a klasifikuje vozidla poměrně s vysokou přesností. Přesto se však při detekci objevovaly nějaké problémy. Mezi jeden z největších problémů patří vozidla ve stínu, kde detektor není schopen všechny takové vozidla najít a označit. Zde je důležitým faktorem barva vozidla a intenzita stínu. Při poměrně světlých barvách vozidel jako jsou bílá a stříbrná jsou vozidla detekována relativně s vysokou přesností, ovšem čím je barva vozidla tmavší, tím má detektor větší problém tato vozidla najít a označit. Tento fakt je znázorněn na obrázku 3.9a, kde jde vidět, že detektor byl schopen označit vozidla s barvami, které jsou více viditelné ve stínu a vozidla s tmavšími barvami již schopen detekovat nebyl.

Jako další problém, který je vhodné zmínit, patří chybná klasifikace objektů, které si jsou z leteckých snímků podobné. S tímto má detektor problém zejména u vozidel typu truck a bus. Toto je zaviněno jejich podobností z leteckých snímků, jelikož oba tyto typy vozidel mají podobný tvar a délku. U tohoto problému je důležité, aby rozlišení, na které detektor mění velikost snímku bylo co nejvíce podobné rozlišení vstupního snímku. Ukázku této chyby je možno vidět na obrázku 3.9b, kde detektor špatně klasifikoval autobus a považuje jej za nákladní vozidlo.

Poslední problémovou částí detekce jsou vozidla, které jdou vidět jen z části. Toto je zapříčiněno snímky z datasetu, kdy byla úmyslně anotována jen taková vozidla, která byla viditelná alespoň z 85%. Toto omezení bylo velice důležité, protože při anotování vozidel, které šly vidět jen z malé části došlo z výraznému zvětšení detekce objektů, které neměly být detekovány. Na obrázku 3.9c lze vidět právě tuto chybu, kde detektor neoznačil vozidla, která jsou z části zakrytá stromy.

(a) Neoznačení vozidla ve stínu (b) Chybná klasifikace (c) Neoznačení části vozidla

Obrázek 3.9: Ukázky problémů detekce

Kapitola 4

Experimenty

V této kapitole budou porovnány výsledky modelů při volbě různých parametrů, při detekci a trénování. Dále bude provedeno porovnání výsledného modelu s jinými architekturami a ověření jeho přesnosti na různých typech snímků.

Pro potřeby otestování modelů je vhodné vytvořit testovací sadu, která bude zahrnovat odlišné snímky za účelem důkladného otestování modelu. K otestování většiny modelů byla použita základní testovací sada, která obsahuje 182 anotovaných snímků, pořízených z různých výšek a z různých zdrojů. Jako jeden z dalších důležitých prvků pro porovnání výsledného modelu je jeho rychlost, která je závislá na kvalitě grafické karty. Pro získání výsledků, které budou níže popsány, byla použita grafická karta Tesla T4.

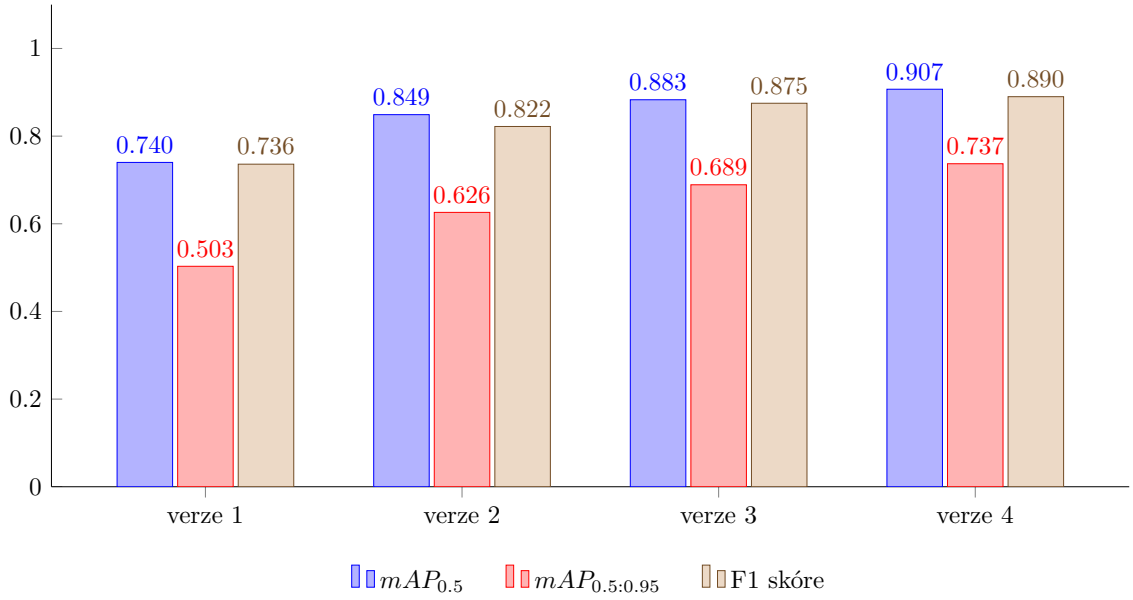
4.1 Porovnání přesnosti modelu podle verze datasetu

K vytvoření dobrého modelu je důležité mít rozsáhlý a rozmanitý dataset. Velikost datasetu určuje kvalitu výsledného modelu, je tedy důležité nasbírat takové množství dat, se kterými model dosahuje vysokých hodnot mAP a zároveň neobsahuje přebytečné množství dat, aby délka trénování nebyla dlouhá. Za účelem dosažení co největší přesnosti bylo postupem času vytvářeno více verzí datasetu, kde nová verze byla vytvořená pokud nastala nějaká výrazná změna v datasetu jako například zvětšení počtu anotovaných snímků nebo aplikování augmentace. Hlavní verze datasetu, které byly během jeho vzniku vytvořeny lze vidět podrobněji popsány v tabulce 4.1.

Tabulka 4.1: Použité verze datasetu

Verze	Rozlišení snímků	Počet snímků	Zdroje	Augmentace
1	různé	∼300	mapy.cz	NE
2	různé	∼500	mapy.cz	NE
3	1280 x 720, 900 x 600	892	mapy.cz a jiné	NE
4	1280 x 720, 900 x 600	1781	mapy.cz a jiné	ANO

Tyto verze nejsou jediné, které byly pro tvorbu výsledného modelu použity, jedná se však o hlavní verze, u kterých došlo k výraznému zlepšení hodnot mAP. Na obrázku 4.1 je vykreslen graf, který zobrazuje hodnoty mAP, precision a recall pro různé verze datasetu, které byly v průběhu vytváření výsledného modelu použity.



Obrázek 4.1: Hodnocení podle velikosti datasetu

Na grafu lze vidět, že se hodnoty mAP a F1-skóre zvětšují v závislosti na verzi datasetu. Toto je z velké části způsobeno zvětšováním počtu snímků, které se ve verzích nacházejí. Největších hodnot mAP dosáhla 4. verze datasetu, která byla zvolena jako finální a je dále použita pro všechny experimenty. Jak již bylo předem zmíněno, velikost datasetu rozhoduje o tom, jak dlouhé trénování finálního modelu může být. Za účelem natrénování modelu s finální verzí datasetu bylo potřeba trénovat síť po dobu 50 hodin. Při porovnání této hodnoty například s 1. verzí datasetu, která obsahovala okolo 300 snímků, je vidět výrazné zrychlení trénování modelu, kdy proces trénování trval pouhých 5 hodin.

4.2 Porovnání různých konfigurací modelů

Jak již bylo předem zmíněno, YOLOv5 obsahuje 4 konfigurace, kde každá z nich má své výhody a nevýhody. Konfigurace jako taková je .yaml soubor, který obsahuje informace pro natrénování sítě jako počet tříd, primární nastavení anchor boxů a jednotlivé vrstvy YOLO modelu. Největším rozdílem těchto konfigurací je poměr rychlosti a přesnosti. Tato skutečnost je zobrazena v tabulce 4.2, ve které lze vidět výsledky konfigurací na základní testovací množině.

Z tabulky je zřejmé, že modely yolov5s a yolov5m patří mezi ty rychlejší, ale méně přesné. Z tohoto důvodu je tyto modely vhodné využít v takových situacích, kdy bychom chtěli model nasadit

Tabulka 4.2: Výsledky různých konfigurací modelu na finální verzi datasetu

Název	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre	Rychlost	GFLOPS	Velikost souboru
YOLOv5s	0.887	0.65	0.867	**10.7 ms**	**16.4**	**14 MB**
YOLOv5m	0.885	0.674	0.862	18.9 ms	50.4	42 MB
YOLOv5l	0.905	0.715	0.889	27.4 ms	114.2	92 MB
YOLOv5x	**0.907**	**0.737**	**0.890**	47.3 ms	217.2	171 MB

do práce s živými přenosy, nebo pro detekce ve videích. Naopak modely využívající konfigurace yolov5l a yolov5x dosahují vyšších hodnot mAP a F1-skóre. Tyto konfigurace není vhodné používat pro detekci ve videích z důvodu jejich rychlosti a vytížení. Dále se v tabulce objevuje nový pojem GFLOPS, což je hodnota udávající počet operací v pohyblivé řádové čárce za sekundu. Tato hodnota se používá jako obvyklé měřítko výpočetní výkonnosti počítačů.

Na obrázků 4.2 lze vidět porovnání detekce na konkrétním případu, kde na obrázku 4.2a jsou vidět detekce vytvořené nejrychlejším modelem s konfigurací YOLOv5s a na obrázku 4.2b detekce vytvořené nejpřesnějším modelem s konfigurací YOLOv5x.

(a) Detekce s konfigurací YOLOv5s

(b) Detekce s konfigurací YOLOv5x

Obrázek 4.2: Porovnání konfigurací YOLOv5s a YOLOv5x

Na obrázku lze vidět, že model YOLOv5x byl schopen detekovat většinu vozidel, oproti modelu YOLOv5s, který některá vozidla schopen označit nebyl. Toto je způsobeno především velikostí modelu, ale kromě toho i tím, že detekce u tohoto modelu mají menší přesnosti a ty u těchto vozidel nespadají nad zvolenou hranici přesnosti.

4.3 Porovnání přesnosti modelu při detekci u snímků z různých výšek

V kapitole 3.7 bylo zmíněno, že vstupní rozlišení, na které se má snímek změnit je velice důležité za účelem dosažení co nejlepších detekcí. Ovšem je také vhodné porovnat přesnost detekce z různých výšek i za předpokladu, že rozlišení na které se mají snímky změnit bude stejné. Největším

problémem u tohoto je kvalita vozidel na snímku, která se větší výškou kvůli stejnému rozlišení zmenšuje.

Pro potřeby ověření této přesnosti byly vytvořeny testovací sady o 50 snímcích pro každou ze zvolených výšek.Zvolené výšky byly rozděleny do tří skupin, a to nízká (∼150 metrů), střední (∼300 metrů) a vysoká (∼600 metrů). Ukázky toho, jak vypadají vozidla z těchto výšek, jsou zobrazeny na obrázku 4.3.

(a) Vozidla z nízké výšky (b) Vozidla ze střední výšky (c) Vozidla z vysoké výšky

Obrázek 4.3: Ukázky viditelnosti vozidel z různých výšek

Výsledky přesnosti detektoru na těchto výškách jsou zobrazeny v tabulce 4.3, ve které jsou zobrazeny hodnoty mAP a F1-skóre pro každou z nich.

Tabulka 4.3: Výsledky detekcí z různých výšek

Název	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre
nízká výška	**0.91**	**0.816**	**0.890**
střední výška	0.828	0.688	0.840
vysoká výška	0.439	0.32	0.512

Z tabulky výše je lehké vyčíst, že čím menší je výška pořízeného snímku, a tedy je jeho kvalita vyšší, tím jsou výsledné detekce přesnější. Největším problémem pro detekce z vyšších výšek je detekce motorek, které na těchto snímcích nebyly téměř vůbec viditelné. Dalším problémem byla již předem zmiňovaná podobnost autobusů a nákladních vozidel, kdy detektor měl problém klasifikovat autobusy a klasifikoval je jako nákladní vozidla.

4.4 Porovnání přesnosti modelu při využití různých optimalizačních funkcí

Optimalizační funkce jsou důležitým prvkem detektorů a je důležité zvolit takovou, se kterou bude detektor dosahovat co nejlepších výsledků. Jak již bylo dříve zmiňováno v kapitole 3.2.2, YOLOv5

obsahuje dvě optimalizační funkce, které mohou být použity při trénování modelu, a to SGD a Adam. Výsledky modelů, které byly trénovány na těchto funkcích jsou zobrazeny v tabulce 4.4, která obsahuje jejich hodnoty mAP na základní trénovací sadě.

Tabulka 4.4: Výsledky různých optimalizačních funkcí na základní testovací sadě

Optimalizační funkce	$mAP_{0.5}$	$mAP_{0.5:0.95}$
SGD	0.907	**0.737**
Adam	**0.938**	0.696

Z tabulky je možno vidět, že model s SGD dosáhl větších hodnot $mAP_{0.5:0.95}$, naopak od modelu s funkcí Adam, u kterého větších hodnot dosáhlo $mAP_{0.5}$. Z této ukázky však není přímo jasné, jaký model zvolit, je tedy důležité modely porovnat hlouběji. Pro další porovnání byla zvolena trénovací sada, která byla použita v kapitole 4.3 pro porovnání modelu na snímcích pořízených z různých výšek. Výsledky získány při použití této testovací sady jsou zobrazeny v tabulce 4.5.

Tabulka 4.5: Výsledky detekcí z různých výšek podle optimalizačních funkcí

	Adam			SGD		
Název	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre
nízka výška	**0.945**	0.74	**0.916**	0.91	**0.816**	0.890
střední výška	**0.881**	0.586	0.793	0.828	**0.688**	**0.840**
vysoká výška	**0.447**	0.262	0.465	0.439	**0.32**	**0.512**

Z výsledků v tabulce lze vidět, že podobně jako u základní testovací sady, tak i zde jsou hodnoty $mAP_{0.5}$ vyšší u modelu s funkcí Adam a hodnoty $mAP_{0.5:0.95}$ vyšší u modelu s funkcí SGD. Dále je vidět, že u modelu s optimalizační funkcí SGD je F1-skóre vyšší u snímků pořízených z výšek 100 a 150 metrů. Z tohoto porovnání je vidět, že oba tyto modely mají své kladné a záporné stránky. Jako finální optimalizační funkce byla zvolena SGD především z důvodu podstatně vyšších hodnot $mAP_{0.5:0.95}$.

4.5 Porovnání přesnosti modelu podle zvolené velikosti snímků při trénování

Jak již bylo zmíněno v kapitole 3.5, zvolení vhodných parametrů trénování modelu je velice důležité. Jedním z těchto parametrů je velikost, na kterou se obrázky musí přeměnit pro potřeby natrénování modelu. Toto je podstatné především z důvodu ztrát na celkové kvalitě snímku, a tak i vyhledávaných objektů a jim specifickým rysům, které jsou pro proces učení sítě klíčové. Hlavním cílem je tedy zvolit takovou velikost, u které dojde při přeměně velikosti snímků k co nejmenším ztrátám na kvalitě. Za účelem porovnání závislosti velikosti snímků při trénování k výsledné přesnosti modelu

byly porovnány velikosti od 416px do 1280px. Výsledky natrénovaných modelů na těchto velikostech lze vidět v tabulce 4.6. Pro testování byla nastavena velikost snímku 1024px.

Tabulka 4.6: Výsledky přesností podle zvolené velikosti snímků při trénování

Zvolené rozlišení	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre
416px	0.511	0.321	0.519
640px	0.875	0.675	0.860
736px	0.899	0.691	0.879
1024px	**0.907**	**0.737**	**0.890**
1280px	0.86	0.664	0.850

Z výsledků výše je vidět, že největších hodnot mAP a F1-skóre dosáhl model, který byl trénován s velikostí 1024px. Toto je zapříčiněno především tím, že obrázky v datasetu mají rozlišení 1280 x 720 a 900 x 600, a proto je velikost 1024 x 1024 nejvíce vhodná. Dále lze vidět, že nejmenších hodnot dosáhl model natrénovaný s velikostí rozlišení 416px, což je způsobeno největší odchylkou od velikostí snímků v datasetu. Dalším důležitým poznatkem je délka trénováni modelu, která se zvětšuje se zvolenou velikostí přeměny snímků. V tomto případě pro dokončení trénování modelu s velikostí 1024px bylo potřeba model trénovat po dobu 50 hodin na rozdíl od velikosti 416px, kde byl model natrénován za pouhých 14 hodin.

4.6 Porovnání přesnosti modelu podle zvolené vstupní velikosti při detekci

Kromě experimentů s velikostí snímků při trénování je vhodné se zaměřit i na testy s různými velikostmi vstupního snímku při detekci. Podobně jako tomu bylo u trénování je důležité, aby vstupní velikost snímku byla co nejbližší jeho reálné velikosti. U nastavování tohoto parametru je podstatné se zamyslet nad výsledným využitím detektoru, protože čím větší je vstupní rozlišení detektoru, tím větší jsou nároky na paměť grafické karty a čas potřebný k vytvoření detekcí. Ukázka výsledků na základní testovací sadě s různými vstupními velikostmi je zobrazena v tabulce 4.7.

Tabulka 4.7: Výsledky modelu pro různá rozlišení detektoru

Vstupní rozlišení	$mAP_{0.5}$	$mAP_{0.5:0.95}$	F1-skóre	Průměrná rychlost detekce
640px	0.792	0.594	0.799	**18.3 ms**
960px	0.883	0.709	0.872	44.0 ms
1024px	0.907	0.737	0.890	45.7 ms
1280px	**0.917**	**0.769**	**0.906**	74.6 ms
2240px	0.756	0.503	0.751	278.2 ms

Z tabulky lze vidět, že největších hodnot mAP a F1-skóre dosáhly detekce při volbě rozlišení 1280px. Toto je zapříčiněno tím, že většina vstupních snímků má rozlišení 1280x720, a tak došlo při změně rozlišení k nejmenším ztrátám na kvalitě. Dále je vidět, že nejmenších přesností dosáhla rozlišení detektoru 640px a 2240px. Na těchto rozlišeních je vidět rozdíl přesnosti detektoru při zvolení příliš malého rozlišení a naopak příliš velkého rozlišení. Největší výhodou při volbě malého rozlišení je výsledná rychlost, kterou detektor potřeboval ke klasifikaci a lokalizaci všech objektů na snímku. Tuto skutečnost lze také vidět v tabulce výše.

4.7 Porovnaní modelů z různých architektur sítí

Za jedno z nejdůležitějších rozhodnutí při tvorbě detektoru je volba vhodné architektury, která bude nejvíce vyhovovat požadavkům na využití detektoru. Je tedy vhodné porovnat architektury a zvolit takovou, která bude objekty detekovat s největší přesností. Pro vytvoření detektoru bylo zvažováno více architektur a je tedy vhodné je krátce popsat a zmínit důvod, proč byly tyto architektury pro konečné porovnání zvoleny. Mezi vybrané architektury k porovnání patří Scaled-YOLOv4 [26], YOLOv3 [9], YOLOv3-tiny [27] a Faster R-CNN [13] jejichž porovnání je popsáno níže.

YOLOv5 – pro porovnání s ostatními architekturami byly zvoleny modely natrénovány na nejmenší konfiguraci YOLOv5s a největší YOLOv5x

Scaled-YOLOv4 – jedná se o rozšíření architektury YOLOv4, která je primárně založena na využití CSPNet umožňující síti měnit její parametry jako hloubku, šířku, rozlišení a strukturu, a přitom dosahovat dobrých rychlostí a přesností

YOLOv3 – jedná se o starší verzi architektury YOLO, podobně jako Scaled-YOLOv4, která byla zvolena především z důvodu její dlouhodobé používanosti

YOLOv3-tiny – tato architektura je komprimovaná verze YOLOv3 za účelem možnosti detekcí a trénování na zařízeních, které mají méně výpočetního výkonu. Tato architektura byla zvolena především z její rychlosti

Faster R-CNN – jako zástupce dvou fázových detektorů byla zvolena architektura Faster R-CNN ve verzi X101-FPN, která dosahovala na datasetu COCO největších hodnot mAP. Pro natrénování této sítě byla použita knihovna Detectron2 [28] vyvíjená firmou Facebook.

Výsledné porovnání těchto architektur je podrobněji popsáno v tabulce 4.8. V této tabulce lze vidět, že nejvyšších hodnot mAP dosáhla architektura YOLOv5x, ale rychlost této architektury je poměrně pomalá oproti jiným ze zvolených architektur. Dále lze vidět, že nejrychlejší architekturou je YOLOv3-tiny, kde průměrný čas, který detektor potřeboval pro nalezení objektů byl pouhých 7ms, zatímco za nejpomalejší se jeví architektura Faster R-CNN, která potřebovala pro vytvoření detekcí průměrně 105ms.

Tabulka 4.8: Výsledky různých architektur na finální verzi datasetu

Název	$mAP_{0.5}$	$mAP_{0.5:0.95}$	Rychlost detekce
YOLOv5x	**0.907**	**0.737**	46.7 ms
YOLOv5s	0.887	0.65	10.7 ms
Scaled-YOLOv4	0.899	0.658	29.1ms
YOLOv3	0.89	0.676	28.4ms
YOLOv3-tiny	0.841	0.553	**7ms**
Faster R-CNN	0.86	0.541	105ms

Na obrázku 4.4 jsou zobrazeny čtyři ukázky detekce na stejném snímku pomocí vybraných výše zmiňovaných architektur.

(a) Detekce s architekturou YOLOv5

(b) Detekce s architekturou Scaled YOLOv4

(c) Detekce s architekturou Faster R-CNN

(d) Detekce s architekturou YOLOv3 tiny

Obrázek 4.4: Porovnání architektur na konkrétním snímku z testovací sady

Kapitola 5

Závěr

Cílem práce bylo implementovat aplikaci schopnou detekovat vozidla z leteckých snímků a popsat teorii s touto problematikou spojenou. Součástí této práce bylo vytvoření aplikace, která umožňuje nastavení parametrů a vytvoření výsledných detekcí na snímcích nebo ve videích.

Ve své práci jsem podrobněji rozepsal teorii potřebnou k pochopení funkčnosti neuronových sítí a detekce v obraze, možnosti řešení a zvolenou architekturu, která byla pro výslednou aplikaci použita. Dále byl popsán proces trénování modelu, výsledné hodnoty, které se využívají pro ohodnocení modelů a problematické části, které se při detekci objevovaly. V práci byly porovnány výsledky modelů s volbou různých parametrů jak při procesu trénování, tak při detekování objektů. Nakonec byly porovnány výsledné hodnoty konečného modelu s jinými architekturami, kde bylo zjištěno, že nejlepších výsledků při detekci vozidel z leteckých snímků dosahovala architektura YOLOv5.

Díky této práci jsem se seznámil principem fungování neuronových sítí a jejich možnostmi využití pro řešení široké škály problémů. Kromě tohoto mi práce dala informace o procesu trénování sítí a zkušenosti s technikami, které se využívají pro problematiku detekce v obraze.

Literatura

1. NASSIF, A. B.; SHAHIN, I.; ATTILI, I.; AZZEH, M.; SHAALAN, K. *Speech Recognition Using Deep Neural Networks: A Systematic Review*. 2019. Dostupné z DOI: 10.1109/ACCESS.2019.2896880.
2. TAYARA, H.; SOO, K. Gil; CHONG, K. T. *Vehicle Detection and Counting in High-Resolution Aerial Images Using Convolutional Regression Neural Network*. 2018. Dostupné z DOI: 10.1109/ACCESS.2017.2782260.
3. MÜLLER, Florian. *Stock Market Prediction Using a Recurrent Neural Network*. 2020-12. Dostupné také z: https://www.relataly.com/stock-market-prediction-using-a-recurrent-neural-network/122/.
4. CHRISLB. Chrislb, 2005-06. Dostupné také z: https://upload.wikimedia.org/wikipedia/commons/6/60/ArtificialNeuronModel_english.png.
5. ULDRICH, Miloš; JURCZYK, Tomáš. [B.r.]. Dostupné také z: https://www.systemonline.cz/clanky/neuronove-site-a-jejich-vyuziti-1.htm.
6. EDUCATION, IBM Cloud. *What are Neural Networks?* [B.r.]. Dostupné také z: https://www.ibm.com/cloud/learn/neural-networks.
7. TRYOLABS. *Object Detection with Deep Learning: The Definitive Guide*. Tryolabs, 2017-08. Dostupné také z: https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/.
8. LIN, Tsung-Yi; GOYAL, Priya; GIRSHICK, Ross B.; HE, Kaiming; DOLLÁR, Piotr. Focal Loss for Dense Object Detection. *CoRR*. 2017, roč. abs/1708.02002. Dostupné z arXiv: 1708.02002.
9. REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. *CoRR*. 2018, roč. abs/1804.02767. Dostupné z arXiv: 1804.02767.
10. NING, Chengcheng; ZHOU, Huajun; SONG, Yan; TANG, Jinhui. *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. Inception Single Shot MultiBox Detector for object detection. 2017. Dostupné z DOI: 10.1109/ICMEW.2017.8026312.

11. GIRSHICK, Ross B.; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. Dostupné z arXiv: 1311.2524.

12. GIRSHICK, Ross B. *Fast R-CNN*. 2015. Dostupné z arXiv: 1504.08083.

13. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross B.; SUN, Jian. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. Dostupné z arXiv: 1506.01497.

14. REDMON, Joseph; DIVVALA, Santosh Kumar; GIRSHICK, Ross B.; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, roč. abs/1506.02640. Dostupné z arXiv: 1506.02640.

15. JOCHER, Glenn; STOKEN, Alex; BOROVEC, Jirka; NANOCODE012; CHRISTOPHERSTAN; CHANGYU, Liu; LAUGHING; TKIANAI; YXNONG; HOGAN, Adam; LORENZOMAMMANA; ALEXWANG1900; CHAURASIA, Ayush; DIACONU, Laurentiu; MARC; WANGHAOYANG0106; ML5AH; DOUG; DURGESH; INGHAM, Francisco; FREDERIK; GUILHEN; COLMAGRO, Adrien; YE, Hu; JACOBSOLAWETZ; POZNANSKI, Jake; FANG, Jiacong; KIM, Junghoon; DOAN, Khiem; YU, Lijun. *ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration*. Zenodo, 2021-01. Ver. v4.0. Dostupné z DOI: 10.5281/zenodo.4418161.

16. WANG, Chien-Yao; LIAO, Hong-Yuan Mark; YEH, I-Hau; WU, Yueh-Hua; CHEN, Ping-Yang; HSIEH, Jun-Wei. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. Dostupné z arXiv: 1911.11929 [cs.CV].

17. LIU, Shu; QI, Lu; QIN, Haifang; SHI, Jianping; JIA, Jiaya. *Path Aggregation Network for Instance Segmentation*. 2018. Dostupné z arXiv: 1803.01534 [cs.CV].

18. LIN, Tsung-Yi; MAIRE, Michael; BELONGIE, Serge; BOURDEV, Lubomir; GIRSHICK, Ross; HAYS, James; PERONA, Pietro; RAMANAN, Deva; ZITNICK, C. Lawrence; DOLLÁR, Piotr. *Microsoft COCO: Common Objects in Context*. 2015. Dostupné z arXiv: 1405.0312 [cs.CV].

19. TZUTALIN. *labelImg* [https://github.com/tzutalin/labelImg]. GitHub, 2015.

20. VAN ROSSUM, Guido; DRAKE, Fred L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

21. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; D'ALCHÉ-BUC,

F.; FOX, E.; GARNETT, R. (ed.). *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, s. 8024–8035. Dostupné také z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

22. NVIDIA; VINGELMANN, Péter; FITZEK, Frank H.P. *CUDA, release: 10.2.89*. 2020. Dostupné také z: https://developer.nvidia.com/cuda-toolkit.

23. CHETLUR, Sharan; WOOLLEY, Cliff; VANDERMERSCH, Philippe; COHEN, Jonathan; TRAN, John; CATANZARO, Bryan; SHELHAMER, Evan. cuDNN: Efficient Primitives for Deep Learning. *CoRR*. 2014, roč. abs/1410.0759. Dostupné z arXiv: 1410.0759.

24. GAD, Ahmed Fawzy. *Mean Average Precision (mAP) Explained*. Paperspace Blog, 2021-04. Dostupné také z: https://blog.paperspace.com/mean-average-precision/.

25. HUI, Jonathan. *mAP (mean Average Precision) for Object Detection*. Medium, 2019-04. Dostupné také z: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

26. WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. Scaled-YOLOv4: Scaling Cross Stage Partial Network. *arXiv preprint arXiv:2011.08036*. 2020.

27. ADARSH, Pranav; RATHI, Pratibha; KUMAR, Manoj. YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In: *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. 2020, s. 687–694. Dostupné z DOI: 10.1109/ICACCS48705.2020.9074315.

28. WU, Yuxin; KIRILLOV, Alexander; MASSA, Francisco; LO, Wan-Yen; GIRSHICK, Ross. *Detectron2* [https://github.com/facebookresearch/detectron2]. 2019.